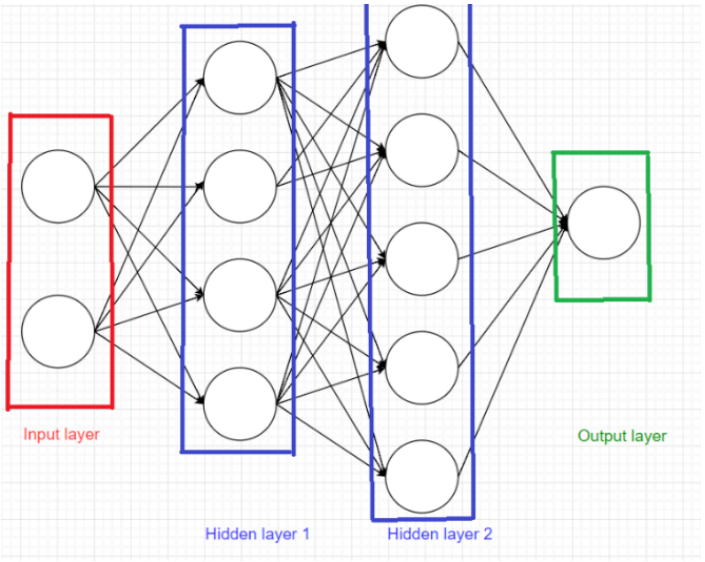
Neural Network

Neural network là một hệ thống tính toán lấy cảm hứng từ sự hoạt động của các nơ-ron trong hệ thần kinh.

Cấu trúc:

Cấu trúc của một Neural Network có thể được biểu diễn như sau:



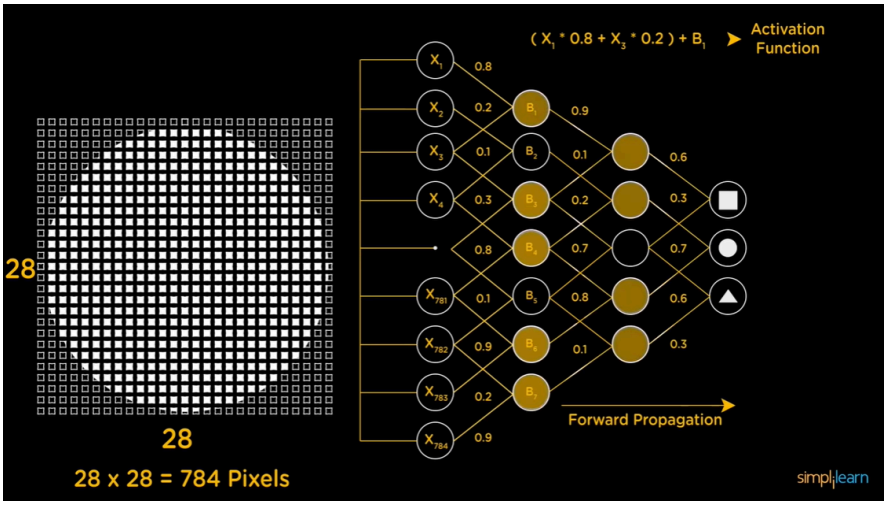
Như vậy đơn giản, ta có thể hiểu Neural Network sẽ có 3 loại layer (tầng chính):

- Input Layer: là tầng đầu tiên, là dữ liệu đầu vào cúa chúng ta. Các hình tròn được gọi là các node, số node trong input layer tương ứng với so feature của dữ liệu.
- Hidden Layers: Có thể có một hay nhiều Hidden Layer, các tầng ẩn ở giữa này có nhiệm vụ trích lọc đặc trưng của dữ liệu đầu vào trong quá trình training. Đây là nơi diễn ra nhiều phép biến đổi và tính toán nhằm tối ưu.
- Output Layer: Dự đoán giá trị đầu ra, nếu trong các bài toán như phân loại, đây thường là một layer softmax hoặc sigmoid. Ví dụ nếu phân loại 8 nhãn thì output layer nó sẽ có 8 node, nhìn chung số node ở output layer tùy thuộc vào mục đích sử dụng.

Quá trình truyền thẳng (Forward Propagation)

Quy ước: một node = 1 neuron

weight gọi tắt là w, bias gọi tắt là b
acitvation function = hàm kích hoạt



Dive sâu vào cấu trúc bên trên hơn ta có: trước tiên ta sẽ quan tâm đến 3 thứ trong neural network: weights (trọng số), bias, và cuối cùng là activation function.

- Weights (ta sẽ ký hiệu là w) là các trọng số, đầu dữ liệu thô đầu vào của chúng ta được đưa vào input layer thứ nhất, sau đó nó sẽ kết nối mới layer tiếp theo thông qua các đường nối, cái này gọi là channel (như trong hình trên thì là các mũi tên màu đen). Mỗi channel này sẽ được gán một giá trị, giá trị đó chính là weights (trọng số).
    - Các trọng số này ban đầu được khởi tạo ngẫu nhiên, và sẽ dần được thay đổi để tối ưu hóa trong quá trình huấn luyện mạng.
    - Sau đó các trọng số này được nhân với các feature đầu vào, nó kiểu như xác định độ quan trọng của feature nào đó với đầu ra.
- Bias (ký hiệu là b): là một giá trị được cộng thêm vào mỗi neural sau khi nhân weight ở trên, bias cũng đóng vai trò quan trọng vì nó sẽ giúp dịch chuyển hướng của hàm kích hoạt (cái này được trình bày rõ hơn trong phần hàm kích hoạt bên dưới).
- Activation function: Hàm kích hoạt: nhìn chung nó là một hàm số để convert giá trị của mỗi neural thành một cái gì đó cho đầu vào của layer tiếp theo. Như trong phần Logistic regression, ta đã biết đến một hàm kích hoạt là sigmoid, để convert giá trị về khoảng 0 1.
    - Ngoài ra nó còn giúp mô hình của chúng ta trở nên phi tuyến hơn, qua đó có thể giải quyết tốt các bài toán phi tuyến. Cái này được trình bày ở phần sau.
    - Nó cũng quyết định xem neural nào được activate, tức là kiểu cái neural đó có thực sự đáng kể với các phần sau hay không.

Tuy nhiên pha cộng bias là tùy vào số neuron ở các lớp sau. Ví dụ ban đầu ảnh là 28x28 tức là input layer là 784, giả sử hidden layer 1 có 16 neuron.

- Mỗi 1 trong 16 node sẽ được nối bởi cả 784 node từ input layer, và mỗi một đường nối (channel) đó lại có 1 weight
- Tức là tổng cái đống đầu tiên đó là 16x784 weights, sau đó đống đó sẽ + 16 cái bias (tương ứng với số node của layer đó)

VD chill: 1 cái phân loại digit, kích thước ảnh là 28x28, có 2 hidden layer đều là 16 node, output layer lúc này sẽ là softmax 10 node:

- Tổng weight lúc này sẽ là 784*16 + 16*16 + 16 *10
- Tổng số bias: 16+16+10

Forward Propagation nhưng mà viết theo toán học:
Đầu vào là vector x:

$$\mathbf{x} = [x_1, x_2, \ldots, x_n]^\top$$

Với n là số feature
Theo như hình, sau khi đưa x vào input layer, ta sẽ nhân feature ở các node với một trọng số w, hay ở đây có thể nói là nhân vector x với ma trận trọng số w, sau đó cộng thêm với vector bias

$$\mathbf{z}^{(1)} = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}$$

(1) ý là chỉ hidden layer thứ nhất.
Sau đó, đưa z(1) qua activation function để convert ra định dạng đầu vào cho layer tiếp theo:

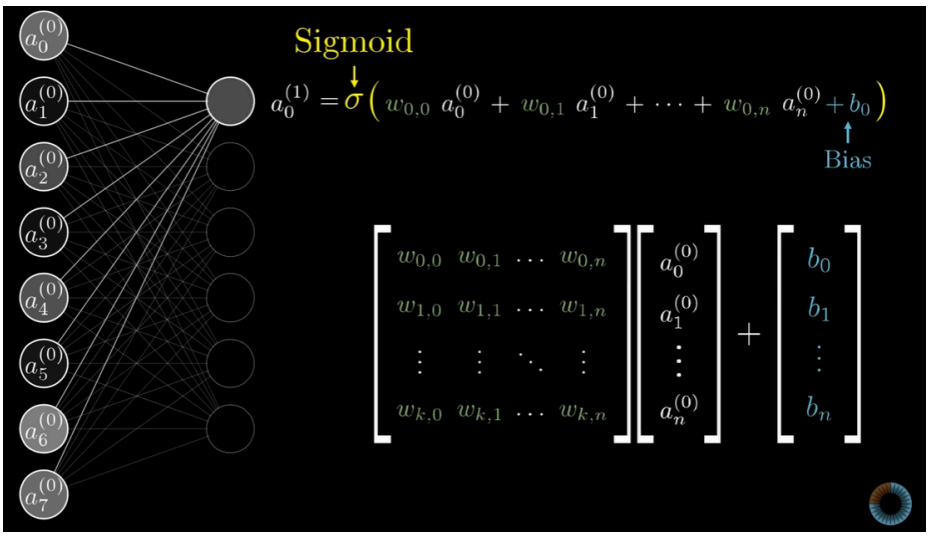
$$\mathbf{a}^{(1)} = f(\mathbf{z}^{(1)})$$

Sau L layer thì nói chung kết quả là:

$$\mathbf{z}^{(L+1)} = \mathbf{W}^{(L+1)}\mathbf{a}^{(L)} + \mathbf{b}^{(L+1)}$$
$$\mathbf{a}^{(L+1)} = g(\mathbf{z}^{(L+1)})$$

Trong đó g là activation function của output layer, thường giống như kiểu softmax hay sigmoid, mục đích g là để tạo định dạng cho đầu ra.

Biểu diễn bằng hình:



Vì giá trị cứ truyền từ lớp đầu sang các lớp tiếp theo rồi đến output như vậy nên nó gọi là quá trình lan truyền thẳng.

Quá trình truyền ngược: Back propagation

Diễn ra sau quá trình truyền thẳng. Sau khi truyền thẳng, ta ra được output predict, việc cần làm bây giờ là so sánh nó với nhãn thực tế và tính loss. Như ta đã biết, mọi mục tiêu của training model đều là tối ưu hóa sao cho loss nhỏ nhất có thể. Tức là trong bài toán này là tìm w và b cho loss nhỏ nhất. **Nói chung mục tiêu của back propagation là update weights và bias.**

Backpropagation bắt đầu từ lớp output và di chuyển ngược lại các lớp trước đó. Mục tiêu là tính toán gradient của hàm mất mát theo từng tham số của mạng neural (weights và biases). Sau đó nó sử dụng chain rule để tính toán đạo hàm dần dần ngược lại. Và việc tính toán được đạo hàm trên các điểm là một yếu tố tốt để có thể tối ưu hóa loss bằng gradient decent.

Chain rule:

$$f(x) = g(h(x))$$

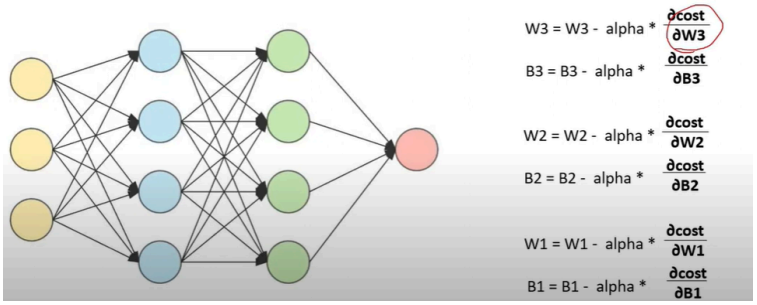
GIả sử g và h đều có thể tính được đạo hàm thì ta có:

$$\frac{df}{dx} = \frac{dg}{dh} \cdot \frac{dh}{dx}$$

Nhìn chung ta sẽ tối ưu hóa hàm loss bằng gradient decent.

weight mới = weight hiện tại - (đạo hàm cost theo weight ở chỗ cần update * l earning_rate).

Phần này sao lại trừ thì là để đi ngược lại hướng đạo hàm tìm global minimum đã được giải thích trong gradient decent.



Trong đó W là weight và B là bias

Để làm được gradient decent này thì ncl cần tính đạo hàm cost theo weight ở chỗ cần update cũng như đạo hàm cost theo b, bằng cách sử dụng chain rule, tính ngược lại từ cost theo weight ở lớp cuối cùng.

Đầu tiên ta giả sử L layer chỉ có 1 neuron
Tính toán đạo hàm C theo w và b

$$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$$

$$a^{(L)} = \sigma(z^{(L)})$$

$$\text{Cost} \longrightarrow C_0(\dots) = (a^{(L)} - y)^2$$

cần hiểu C thay đổi như thế nào nếu W thay đổi nhẹ, tuy nhiên theo công thức có thể thấy, sự thay đổi của w có thể kéo theo sự thay đổi của cả z và a
Vậy ta có thể viết thành một chain rule:

$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C0}{\partial a^{(L)}}$$

Và ta có thể dễ dàng tính đạo hàm của các phần nhỏ bên vế phải

$$\frac{\partial C0}{\partial a^{(L)}} = 2(a^{(L)} - y)$$

$$\frac{\partial a^{(L)}}{\partial z^{(L)}} = \sigma'(z^{(L)})$$

$$\frac{\partial z^{(L)}}{\partial w^{(L)}} = a^{(L-1)}$$

Tuy nhiên đây là ví dụ cho 1 training sample. Tổng quá hóa lên cho tất cả các training sample:

$$\frac{\partial C}{\partial w^{(L)}} = \frac{1}{n}\sum_{k=0}^{n-1}\frac{\partial C_k}{\partial w^{(L)}}$$

với k là số training sample

Với b thì cũng tương tự, nhưng khi thay b vào, đạo hàm nó bằng 1 luôn nên dễ hơn

$$\frac{\partial C_0}{\partial b^{(L)}} = \frac{\partial z^{(L)}}{\partial b^{(L)}}\frac{\partial a^{(L)}}{\partial z^{(L)}}\frac{\partial C0}{\partial a^{(L)}} = 1\sigma'(z^{(L)})2(a^{(L)} - y)$$

Ta cũng có thể thấy cost cũng nhạy cảm với cả activation của các lớp trước đó, nên đây chính là ý tưởng cho việc back propagation ra đời.

$$\frac{\partial C_0}{\partial a^{(L-1)}} = \frac{\partial z^{(L)}}{\partial a^{(L-1)}}\frac{\partial a^{(L)}}{\partial z^{(L)}}\frac{\partial C0}{\partial a^{(L)}} = w^{(L)}\sigma'(z^{(L)})2(a^{(L)} - y)$$

Tổng quát hóa lên nhiều neuron thì thêm index

$$\frac{\partial C_0}{\partial a_k^{(L-1)}} = \sum_{j=0}^{n_L-1} \frac{\partial z_j^{(L)}}{\partial a_k^{(L-1)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial C_0}{\partial a_j^{(L)}}$$

$$\frac{\partial C}{\partial w_{jk}^{(l)}} = a_k^{(l-1)} \sigma'(z_j^{(l)}) \frac{\partial C}{\partial a_j^{(l)}}$$